VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2015

Lukáš Raška

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Raška**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Kožusznik, Ph.D.**

Konzultant bakalářské práce: Ing. Daniel Frejek

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Rád bych poděkoval Danielu Frejkovi, Lukáši Dvorokovi, Martinu Macečkovi a Paulu Ahlskogovi za možnost pracovat na tomto zajímavém projektu a bez kterých by tato práce nevznikla.

Abstrakt

Cílem bakalářské práce bylo absolvování odborné praxe ve firmě Tieto Czech s.r.o. za účelem vytvoření funkčního produktu pro uchovávání dat s pomocí open-source technologií, jakožto náhradu za placené technologie (Atlassian Confluence, Microsoft SharePoint) a sjednocení aktuálně používaných technologií napříč firmou. Práce popisuje teoretickou část i praktickou část implementace open-source produktu XWiki do korporátního prostředí a vytvoření vysoce dostupné platformy.

Klíčová slova: XWiki, bakalářská práce, Tieto, aplikační cluster, vysoká dostupnost, Java, databáze, rozkládání zátěže, CMS

Abstract

The aim of this bachelor's thesis was to absolve individual professional practice in Tieto Czech s.r.o. company for the purpose of creating company-wide content management platform based on open-source technologies as a replacement for paid proprietary technogies (such as Atlassian Confluence or Microsoft SharePoint) and unification of currently used technologies. Thesis describes theoretical and practical implementation of open-source product XWiki into corporate environment and creation of high-available cluster platform.

Keywords: XWiki, bachelor's thesis, Tieto, application cluster, high-availability, Java, databases, load-balancing, CMS

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
APR	–	Apache Portable Runtime
BIO	–	Blocking Input/Output
CDI	–	Contexts and Dependency Injection
CI	–	Continuous Integration
CMS	–	Content Management System
GARBD	–	Galera arbitrator daemon
GSSAPI	–	Generic Security Service Application Program Interface
HA	–	High availability, vysoká dostupnost
NAS	–	Network Attached Storage
HTML	–	HyperText Markup Language
I/O	–	Input / Output
J2EE	–	Java 2 Enterprise Edition
JAAS	–	Java Authentication and Authorization Service
JDBC	–	Java Database Connectivity
JNDI	–	Java Naming and Directory Interface
JPA	–	Java Persistence API
JSR	–	Java Specification Request
JVM	–	Java Virtual Machine
KDC	–	Key Distribution Center
LDAP	–	Lightweight Directory Access Protocol
MVCC	–	Multiversion concurrency control
NIO	–	Non-blocking Input/Output
NTLM	–	NT LAN Manager
PXC	–	Percona XtraDB Cluster
RHCS	–	Red Hat Cluster Suite
SaaS	–	Software as a service
SPNEGO	–	Simple and Protected GSSAPI Negotiation Mechanism
SSO	–	Single sign-on
STONITH	–	Shoot The Other Node In The Head
URL	–	Uniform Resource Locator

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce popisuje průběh implementace open-source platformy XWiki jakožto náhradu za současně používaná proprietární řešení a vytvoření jednotné databáze interních dat, jakožto i interních dat jednotlivých týmů. XWiki bylo vybráno z důvodu již existující implementace v rámci našeho týmu (Java middleware), kde je daná platforma používána pro uchovávání interních informací o zákaznících a o týmu samotném.

Práce je rozdělena na dvě části, teoretickou a praktickou. Úkolem teoretické části je popsat platformu XWiki a možnosti využití ve firemním prostředí s přihlédnutí k bezpečnostním hlediskům. Praktická část popisuje vytvoření služby (SaaS) pro celofiremní použití s více jak 10 tisíci aktivními uživateli.

2 Profil firmy

2.1 Tieto

Tieto je finsko-švédská IT firma zabývající se poskytováním IT služeb širokému spektru zákazníků v soukromém i veřejném sektoru. Firma založená v roce 1968 s hlavním sídlem v Helsinkách zaměstnává téměř 14 000 expertů z oblasti informačních technologií.

V České republice se nachází od roku 2004 pobočka Tieto Czech s.r.o. s centrem v Ostravě. Zde má Tieto více než 2000 zaměstnanců a je tak třetí největší pobočkou Tieto, jakožto i jedním z největších IT zaměstnavatelů v celé republice. Kromě hlavního sídla Tieto Towers v Ostravě Mariánských horách má Tieto Czech s.r.o. pobočky ve Vědecko-technickém parku Ostrava a v Brně.

2.2 Pozice ve firmě

Ve firmě jsem byl členem Java týmu pod vedením Ing. Daniela Frejka, zabývajícím se poskytováním podpory pro middleware aplikace přímo i nepřímo souvisejících s Java technologiemi. Tým se dělí na tým poskytující *continuous services*[1] a dva projektové týmy. Tým zabývající se Oracle technologiemi (především WebLogic aplikační server) a tým zabývající se IBM a open-source technologiemi, jehož součástí jsem byl i já. Zde jsem se specializoval především na Tomcat aplikační server, Nginx a Apache httpd webové servery a přidružené technologie. Mým hlavním úkolem bylo vytvořit zmíněnou platformu, vedlejšími úkoly byla instalace a správa zákaznických serverů a investigace a řešení vzniklých či možných problémů.

[1] Nepřetržité udržování zákaznických prostředí

3 XWiki

XWiki je systém pro dynamickou správu obsahu (CMS), označovaná jako wiki druhé generace. Principem *wiki první generace* je sdílení především textového obsahu, wiki druhé generace slouží nejen pro sdílení dat, ale i jako platforma pro jednoduché vytváření webových aplikací. XWiki vytvořil Ludovic Dubost v roce 2003 a v roce 2006 převzal pozici šéfa vývoje Vincent Massol, dlouhodobý open-source vývojář a expert na Maven, JUnit a agilní metody vývoje.

Z diskuze o nově vytvářeném celofiremním řešení lze vytvořit seznam požadavků, kterými by platforma měla disponovat.

- Sdílení textově orientovaných dat napříč firmou
- Obsah může tvořit jakýkoliv zaměstnanec
- Správa oprávnění (určité kategorie může modifikovat pouze určená skupina lidí)
- Propojení s interními systémy (automatické přihlášení, interní sociální služba, aj.)
- Responzivní design
- Možnost vytváření oddělených celků pro potřeby týmů
- Snadná rozšiřitelnost, správa a nasazení
- Možnost lokalizace do více jazyků
- Správa verzí stránek a příloh

Nejčastěji používané open-source systémy neumožňují snadnou tvorbu webových aplikací koncovými uživateli či dynamickou správu oprávnění. Pro tyto účely se výborně hodí právě navrhovaná XWiki. Hlavním principem XWiki jsou stránky. Každá takováto stránka patři do unikátní kategorie (dále označována jako *Space*) a musí mít unikátní název (obsažený v URL, taktéž obsahuje zobrazovaný titulek). Také může obsahovat libovolné množství příloh. Stránka má nejvýše jednoho rodiče a neomezené množství potomků, lze tedy tvořit obsah ve stromové struktuře. Každá stránka může obsahovat kromě své textové části i objekty, či být třídou. Objektem může být interní struktura, nebo jiná stránka. Pro každou stránku a Space lze nastavit oprávnění pro uživatele a skupiny.

Jelikož základním stavebním prvkem XWiki jsou stránky, každý registrovaný uživatel je veden jako stránka obsahující objekt *XWiki.XWikiUsers* s příslušnými parametry. Analogicky je také každé nastavené oprávnění či komentář objektem přiřazeným dané stránce. Z tohoto důvodu obsahuje každý Space stránku **WebPreferences**, která jej definuje, a stránku **WebHome**, která je jeho hlavní stránkou.

XWiki obsahuje hlavní wiki (označená xwiki), volitelně lze vytvářet další wiki (zvané subwiki) s oddělenou správou. Subwiki může obsahovat lokální uživatele, nedostupné na hlavní wiki a má oddělenou administraci, databázi i URL, lze ji tedy efektivně použít pro potřeby týmů.

3.1 Rozšiřování

Důležitou funkcionalitou systému je jeho snadná rozšiřitelnost. XWiki nabízí dvě možnosti rozšíření: **komponenty**[1] a **rozšíření**.

Komponenty jsou Java knihovny globálně rozšiřující stávající funkcionalitu systému a jsou načteny při startu platformy. Komponenta může mít libovolné použití, od REST API, přes WYSIWYG editor, API pro odesílání emailů, až po LDAP integraci a SSO autentizaci.

Rozšíření (anglicky *extensions*) jsou rozšíření vytvářená přes webové rozhraní. Mohou být klientská (JavaScript a CSS pomocí *StyleSheetExtension* a *JavascriptExtension* tříd[2]), či serverová[3]. Zde lze díky integraci JSR-223 použít množství skriptovacích jazyků. Primárně se zde používá šablonovací systém Velocity, které umožňuje integraci se systémem oprávnění. Díky tomu můžou této funkcionality využívat všichni registrovaní uživatelé bez nutnosti řešit problematiku nedovoleného přístupu k datům. Běžné skriptovací jazyky (např. Groovy či Python) tuto integraci neumožňují, je zde tedy definováno programovací oprávnění, které umožní přistup k celému Java API platformy. Dále lze tvořit rozšíření pomocí maker, která umožňují znovupoužitelnost určité funkcionality (od načtení obsahu jiné stránky po vykreslení struktury celé wiki).

3.2 App Within Minutes

Obrovskou výhodou XWiki je možnost tvoření aplikací uživateli bez znalosti programování. App Within Minutes je komponenta, která umožňuje jednoduše vytvořit definici XWiki třídy a všechny potřebné stránky s uživatelsky příjemným prostředím. Výsledkem je stránka, kde může uživatel přidávat záznamy s jasně definovanou strukturou.

V prvním kroku se definuje název vytvořené aplikace a její popis. Ve druhém kroku si uživatel pomocí principu *Drag and Drop* vybere, která políčka má daný záznam obsahovat. Zde má uživatel možnost zvolit například textové pole, seznam, či pole s výběrem uživatelů. V posledním kroku si uživatel vybere, která políčka záznamů chce zobrazovat na hlavní stránce aplikace. Vytvořená struktura obsahuje dva Space, první obsahuje definici třídy a jak se budou záznamy zobrazovat (technická část), druhý obsahuje vytvořené záznamy (každý záznam je jedna stránka s objektem vytvořené třídy).

Představme si tedy že existuje subwiki využívána týmem vývojářů využívajícího agilního přístupu k vývoji Scrum. Scrum master by potřeboval mít přehled o vývoji nových funkcionalit (*Scrum task board*). Zde by stačilo ve druhém kroku vytvořit aplikaci s následujícími políčky:

1. Název funkcionality (text)
2. Detailní popis funkcionality (dlouhý text)
3. Vývojář (uživatel, vícenásobná volba)
4. Sprint (seznam)
5. Stav (výčet)

V případě programování dané funkcionality by byl třeba vývojář a určité množství času (peněz). V případě použití komponenty App Within Minutes může funkcionalitu vytvořit kdokoliv v krátkém čase.

3.3 Technické zázemí

XWiki je J2EE aplikace používající moderní technologie webových aplikací.

Z hlediska klientské části využívá knihovny Bootstrap pro dosažení moderního responzivního layoutu. V serverové části XWiki využívá technologie *Servlet 3.0* a *Apache Struts* pro vykreslování. Na datové vrstvě se používá JPA, konkrétně implementace Hibernate, JDBC a DBCP pro *connection pooling*[2]. Pro logování se využívá knihovny SLF4J. Pro cache mezi datovou a aplikační vrstvou se využívá Infinispan. Pro *cluster-awareness*[3] je využito knihovny JGroups. Díky využití technologie Java Beans je možné nahradit knihovny Infinispan a JGroups za libovolnou implementaci (např. OSCache či HazelCast). Pro usnadnění vývoje a znovupoužitelnosti instancí tříd je využíváno CDI. Z vývojového hlediska je využíváno nástroje Maven pro automatickou správu závislostí a build procesu.

Pro zpracování kancelářských je použit OpenOffice spuštěný jako *headless*[4] server. Pro vyhledávání je vyžíván Apache Solr, který pro použití v clusteru využívá Apache ZooKeeper z rodiny Hadoop pro uchování interní konfigurace.

V implementovaném prostředí je místo DPCB použit HikariCP z důvodu rychlosti. Pro perzistentní uložení dat je použita databáze Percona, rychlejší modifikace MySQL. Místo OpenOffice je použit LibreOffice, jakožto více vyvíjená alternativa.

[2] Vícenásobné připojení do databáze pro sdílení spojení
[3] Aplikace má podvědomí o ostatních instancích
[4] Běžící na pozadí

4 Teoretická část

4.1 Rozbor problémů

Při plánování nového prostředí bylo třeba zdokumentovat možné problémy a navrhnout infrastrukturu s ohledem na možné špičkové zátěže, jakožto i celodenní dostupnost.

Celý životní cyklus implementace lze rozdělit do několika částí

- Analýza starého prostředí
- Návrh infrastruktury nového prostředí s přihlédnutím k vysoké dostupnosti
- Instalace a konfigurace nového prostředí
- Modifikace platformy pro použití ve firemním prostředí (implementace SSO, programování dodatečných funkcionalit)
- Testování nového prostředí
- Poloautomatická migrace z jiných systémů

4.2 Staré prostředí

Staré prostředí se skládá ze dvou serverů pro dosažení vysoké dostupnosti. Zde byla možnost postavit *cluster* jako *active-passive*, tedy systém, kdy je aktivní pouze jeden server a v případě výpadku převezme jeho funkci server druhý. Z důvodu zbytečného plýtvání prostředků byla tato možnost zavrhnuta a cluster je postaven jako *active-active*, tedy oba servery jsou aktivní ve stejnou dobu a zátěž je rozkládána.

Pro dosažení skutečné vysoké dostupnosti je možné použít *clusterware* (software spravující cluster), který periodicky kontroluje stav clusteru a spravuje běžící služby. V korporátní sféře jsou nejpoužívanější dvě služby, *Veritas Cluster* a *RHCS*. Oba produkty jsou placené, nicméně RHCS je postaven na open-source projektu **Linux-HA**, volba tedy padla na něj. Clusterware se skládá ze dvou služeb *Corosync* a *Pacemaker* (ve starší verzi Heartbeat a Pacemaker). Corosync se stará o stav clusteru a zajišťuje synchronizaci dat mezi servery v clusteru. Pacemaker se stará o jednotlivé služby na serverech a hlídá jejich stav. Zde je možnost nastavit služby jako active-passive, nebo jako *cloned*. Active-passive znamená, že služba beží pouze na jednom serveru v clusteru a v případě výpadku daného serveru bude služba automaticky migrována na server jiný. Cloned služba funguje jako active-active, tedy služba běží na všech serverech v clusteru. Pro potřeby platformy je použita kombinace obou služeb.

Vzhledem k použití active-active clusteru běží všechny aplikace v cloned módu, tedy na obou serverech. Active-passive je použito na *load balancing*[5]. To je možné zpřístupnit několika způsoby. První možnost je použití více A záznamů v DNS. V případě návštěvy stránky bude posíláno více IP adres a systém vybere vždy jednu, na kterou se připojí. Zde vzniká řada bezpečnostních problémů, nejdůležitější je zveřejnění IP adres jednotlivých

[5] Rozkádání zátěže

serverů. Pro vyhnutí se těmto problémům je vhodné použít softwarový loadbalancer v kombinaci s plovoucí IP adresou. Zde figuruje IP adresa, která je aktivní pouze na jednom serveru v daném čase (active-passive) a v případě výpadku serveru je automaticky přesunuta na jiný fungující server. V tomto případě tedy active-passive služby budou plovoucí IP adresa a SW loadbalancer nginx.

Pro vysokou dostupnost je třeba nastavit i další služby. Službu ZooKepeer, která je použita pro uchovávání konfigurace pro službu Solr, lze použít v cluster módu. Lze tedy nastavit komunikaci mezi všemi aktivními instancemi v clusteru. V případě výpadku webového serveru není třeba řešit výpadek ostatních služeb, nicméně v případě výpadku služby ZooKeeper je třeba mít k dispozici náhradu, ze které si bude moci Solr načíst aktuální konfiguraci. Pokud zde nastavíme Solr pro více ZooKeeper serverů, docílíme plné vysoké dostupnosti.

Nakonec zde vzniká problém s duplikací databáze. Vzhledem k vybranému režimu active-active je nejvhodnější databázi nastavit pro multi-master replikaci, tedy v případě zápisu na jakýkoliv server bude změna viditelná v celém clusteru. V případě použití *Percona XtraDB Cluster* je tento způsob replikace zajištěn s pomoci knihovny Galera a všechny změny jsou replikovány skrze celý cluster. Nicméně pokud použijeme pouze 2 servery, vzniká zde problém zvaný *split-brain*. V tomto případě, pokud vznikne síťový problém mezi dvěma fungujícími servery, je komunikace přerušena a aktivní servery netuší, který z nich má aktuální data. Pro vyhnutí se možné ztrátě dat zde funguje politika STONITH, neboli aktivní server, který netuší, zda je aktivní, se přeruší v případě kolize. Pokud se server vypne a spustí, nevzniká zde tento problém (spuštěný server ví, že nemusí mít aktuální data).

V případě síťového problému se oba servery vypnou a databáze není dostupná. Toto je důvod, proč všechny clusterové konfigurace existují v alespoň třech serverech. Pokud máme k dispozici pouze dva servery, je vhodné využít *Garbd*. Garbd funguje jako daemon suplující existující databázový server, ale neuchovává data. V případě výpadku jednoho serveru se přikloní na stranu dostupného serveru a vynutí synchronizaci serveru nedostupného. Zde je třeba brát v úvahu nedostupnost konkrétní sítě, je tedy doporučeno mít třetí server (či Garbd) na oddělené síti. V případe vzniklého problému split-brain je třeba spustit jeden server jako hlavní, pro Perconu zde figuruje init parametr *bootstrap-pxc*. Pokud dojde ke kompletnímu výpadku služby (vypnutí všech serverů v jednu dobu) není třeba split-brain řešit díky direktivě *pc.recovery*[4], kdy je stav clusteru uložen na disk.

V případě clusterování služby je také třeba brát v úvahu dostupnost uložených dat. U starého prostředí je použit GlusterFS, jakožto distribuované řešení, všechna data jsou tedy dostupná na všech serverech. Není zde použit *sharding*, jelikož lze předpokládat nedostupnost více serverů.

Samotná XWiki aplikace beží pod aplikačním serverem Tomcat 7 společně s vyhledávací aplikací Solr.

4.3 Nové prostředí

Při návrhu infrastruktury nového prostředí bylo třeba v první řadě brát v úvahu vyšší návštěvnost a zátěž platformy. Následně bylo třeba zapřemýšlet nad distribuovaným da-

tovým úložištěm. Použité GlusterFS bylo jistou variantou, nicméně vyžaduje N-násobný počet místa (N = počet serverů v clusteru). Nakonec bylo zvoleno síťové úložiště (NAS) nabízené jako interní služba. Celkový pohled na infrastrukturu je zobrazen na obrázku č. 2.

Pro eliminaci co nejvíce problémů byla zvolena následující počáteční konfigurace:

- 3 virtuální servery
  - 8 procesorů
  - 16 GB operační paměti
  - 40 GB disk na operační systém
  - 30 GB disk na aplikace
  - NAS storage připojený přes protokol CIFS
  - Red Hat Enterprise Linux 6.5
- Active-active cluster (Linux-HA projekt)
  - OpenAIS stack
  - Databáze (cloned)
  - LibreOffice (cloned)
  - ZooKeeper (cloned)
  - Aplikační server Tomcat (cloned)
  - Plovoucí IP adresa + load balancer Nginx (active-passive)
- Oracle JDK 7 jakožto stabilní otestované JVM
- Apache Tomcat 7
- nginx 1.6 webový server pro load-balancing
- Vyhledávací služba Apache Solr 4.9
- ZooKeeper 3.4.6
- LibreOffice 4.3
- Percona XtraDB Cluster 5.6
- XWiki Enterprise 6.2
- Munin jako monitorovací nástroj

Vzhledem k vestavěné podpoře plovoucí IP adresy (VRRP protokol) v clusterware není třeba využívat dalšího software (například Keepalived).

Důvodem pro zvolení řešení Percona XtraDB Cluster je zabudována podpora multi-master replikace a rychlejší zpracování požadavků vzhledem k MySQL. Percona XtraDB

je *drop-in*[6] náhrada transakčního úložiště InnoDB s mnoha výkonnostními vylepšeními a plnou podporou nejpoužívanější metody verzování MVCC. Percona dále také nabízí nástroje pro správu databáze, například *innobackupex* pro neblokující zálohu databáze.

Jakožto aplikační server byl zvolen Apache Tomcat, open-source aplikační J2EE server. XWiki nevyžaduje standard EJB3, je tedy možné použít právě Tomcat jakožto jednoduché a rychlé řešení. Oproti aplikačnímu serveru Jetty nabízí více možností modifikace a výkonnostních úprav. Pro urychlení odbavení požadavků byly použity *Tomcat-Natives*, nativní systémové konektory s pomocí APR. Webový server nginx byl zvolen jakožto load-balancer a proxy server pro odbavení statických dat a SSL. Pro load-balancing nabízí nginx několik možností konfigurace:

- *Round-robin* - každý následující požadavek je předán následujícímu dostupnému serveru
- *Least-connected* - příchozí požadavek je předán dostupnému serveru s nejméně aktivními spojeními
- *IP-hash* - na IP adresu klienta je použita hashovací funkce, která určuje číslo serveru

Nginx nabízí i další load-balancing funkcionalitu zvanou *sticky cookies*, která pracuje na principu sticky-sessions, nicméně pouze v placené verzi. Tato funkcionalita při první návštěvě nastaví klientovi zašifrovanou cookie, která určuje, na který server se připojil naposled a v případě následující návštěvy je klient směrován primárně na tento server, pokud je dostupný. Toto efektivně řeší problémy se synchronizacemi sezení (session), uživatel tedy nebude například náhodně odhlašován. Při použití round-robin či least-connected by každý požadavek mohl být směrován na rozdílný server, nebyla by tedy zajištěna integrita požadavků (například v případě omezené sekce v informačním systému). Při použití IP-hash zde tento problém nevzniká, nicméně při použití v monolitické síti (interní síťový subnet) dochází k přetěžování pouze jednoho serveru, což se může negativně projevit na době odezvy.

Přestože neplacená verze nginx serveru funkcionalitu sticky cookies nenabízí, existuje opensource modul *nginx-sticky-module-ng*, který je ale třeba explicitně nastavit při kompilaci aplikace. Naštěstí je nginx distibuován s tzv. *specfile*, neboli souborem s definicemi pro snadné vytvoření RPM balíčku. Je tedy možné vytvořit instalační balíček s potřebným modulem a využívat schopnosti balíčkovacího systému. Nginx dále nabízí i možnost použít experimentální protokol SPDY, který je základem nového protokolu *HTTP/2*.

4.3.1 Výkonnostní testy zvolených technologií

Výkonnostní testy byly provedeny na následující konfiguraci:

- Čistá minimalistická instalace Debian 7.7 Wheezy ve virtualizaci LXC
- Sdílený procesor Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz, 2 jádra s HT

[6] vzájemně kompatibilní

Aplikační server	Průměrná doba (ms)	Nejdelší požadavek (ms)	Propustnost (req/s)
Tomcat 7 (BIO)	10.603	1331	14052.56
Tomcat 7 (NIO)	13.299	1072	11204.04
Tomcat 7 (APR)	10.132	307	14705.87
nginx 1.2.1	3.862	221	38578.34

Tabulka 1: Výkonnostní test, 100 000 požadavků, 150 požadavků v jednu dobu, prázdná stránka, s Keepalive funkcionalitou

Aplikační server	Průměrná doba (ms)	Nejdelší požadavek (ms)	Propustnost (req/s)
Tomcat 7 (BIO)	16.802	1024	8868.14
Tomcat 7 (NIO)	21.505	3022	6928.57
Tomcat 7 (APR)	14.745	1014	10105.07
nginx 1.2.1	13.213	39	11276.59

Tabulka 2: Výkonnostní test, 100 000 požadavků, 150 požadavků v jednu dobu, prázdná stránka, bez Keepalive funkcionality

- SSD Corsair Force 3

Pro otestování byl použit Apache Benchmark. Každý test byl puštěný třikrát a výsledky byly zprůměrovány. Tento test nereflektuje schopnost technologií na použitých virtuálních serverech, jde spíše o srovnání použitých aplikačních serverů. Z výsledků testů (tabulky 1, 2, 3, 4, 5) lze vyvodit, že nejlepší bude použít kombinaci nginx a Tomcat s APR konektorem. Nginx nabízí nejstabilnější a nejrychlejší odbavení požadavků, je tedy jej vhodné použít pro statická data a jako proxy server s load-balancingem. Nginx navíc nabízí rozšířené možnosti cache (microcache pro proxy) a odolnost vůči rozličným útokům (například Slowloris) bez nutnosti instalace dodatečných modulů.

Tabulka č.1 ukazuje zátěžový test s konfigurací *ab -k -n 100000 -c 149 [URL]*. Zde je viditelné, že nginx je bezkonkurenčně nejrychlejší. U Tomcat konektorů lze vidět, že rozdíl mezi blokujícími a nativními konektory je zanedbatelný, což je pravděpodobně způsobeno nezatíženým procesorem (odbavování čisté stránky, žádné I/O), nicméně stabilita nativních konektorů (rozdíl mezi průměrnou dobou a nejdelší dobou) je mnohem lepší při použití APR. Tento způsob testu nám dokáže ukázat chování aplikačních serverů při zatížení jedním strojem díky použitému Keepalive (přepínač *-k*).

Následující testy, reprezentované tabulkami č. 2 a č. 3, byly spuštěny bez Keepalive a dokážou nám ukázat předpokládanou odezvu při špičkové zátěži (pouze statická data). Rozdíl mezi Tomcat konektory a nginx serverem se výrazně zmenšil, nicméně zde všude dominuje právě nginx a APR konektory. V testu č. 4 lze vidět razantní nárust odezvy při použití Java SSL implementace (NIO, BIO) oproti OpenSSL (nginx, APR). Z tohoto důvodu je nginx použit pro odbavování statických dat a zpracování SSL s podporou microcache pro proxy požadavky.

Aplikační server	Průměrná doba (ms)	Nejdelší požadavek (ms)	Propustnost (req/s)
Tomcat 7 (BIO)	33.240	7044	4482.56
Tomcat 7 (NIO)	31.932	1120	4666.17
Tomcat 7 (APR)	27.257	1097	5466.40
nginx 1.2.1	22.041	1027	6760.06

Tabulka 3: Výkonnostní test, 100 000 požadavků, 150 požadavků v jednu dobu, soubor o velikosti 100 kB, bez Keepalive funkcionality

Aplikační server	Průměrná doba (ms)	Nejdelší požadavek (ms)	Propustnost (req/s)
Tomcat 7 (BIO)	306.199	1258	32.66
Tomcat 7 (NIO)	303.424	1153	32.96
Tomcat 7 (APR)	21.664	65	461.59
nginx 1.2.1	19.629	47	509.45

Tabulka 4: Výkonnostní test SSL, 1000 požadavků, 10 požadavků v jednu dobu, prázdná stránka, bez Keepalive funkcionality

4.4 Verzování konfigurací a plánování vývoje

Pro dokumentování modifikací operačního systému byla použita utilita *etckeeper*, která vytvoří repozitář na konfiguračním adresáři */etc* pomocí volitelného verzovacího systému (zde Git). Etckeeper také podporuje integraci s balíčkovacím systémem *yum*, nedovolí tedy nainstalovat balíčky bez zdokumentovaných předchozích úprav.

Konfigurace celé webové aplikace byla uložena do Git repozitáře na interní instanci aplikace *GitLab*. Zde jsou také uložené veškeré vyvíjené komponenty a *Web Hooks* jsou použité pro integraci s CI systémem Jenkins. Každá aktualizace komponenty tedy spustí automatickou kompilaci celé komponenty, JUnit testy a kontrolu kompatibility. Pro komplexnější projekt by bylo vhodné zvážit použití centralizovaného úložiště, například *Artifactory*.

Pro efektivní metodu vývoje bylo zváženo několik metodik. Vzhledem k již vyvinutému software nebyl důvod rozebírat výhody a nevýhody agilního přístupu při vývoji celého informačního systému. Z důvodu malého týmu byla zvolena modifikace agilního Scrum vývoje. Vývoj jednotlivých komponent probíhal v iteracích, vždy byl vytvořen funkční celek a další funkcionalita byla vyvíjena dodatečně. Zde bylo využíváno automatické správy závislostí a kompilace pomocí aplikace Maven. Dále zde byl používán *Checkstyle plugin* pro dodržování určitého programovacího standardu.

Základem programovacího standardu jsou následující položky:

- Javadoc dokumentace
- Odsazení (nahrazení tabulátorů mezerami)
- Maximální komplexnost metod (cyclomatic complexity - maximální vnořenost bloků, délka metody)

- Dodržování XWiki code style

4.5 Monitoring

Pro základní systémové monitorování je používána interní služba v rámci celofiremní integrace. Pro podrobnější sběr dat pro potřeby výkonnostního zlepšování platformy byl zvolen *Munin*. Munin je založený na *RRDTool*, databázi pro ukládání časově závislých dat a generování grafů, a byl vybrán z důvodu jednoduchosti a snadné rozšiřitelnosti monitoringu.

Munin je napsán v programovacím jazyce Perl a je složen ze dvou částí: **Munin server** a **Munin agent**. Munin server obsahuje tři nástroje pro generování grafů, HTML stránek a sběr dat. Munin agent slouží jako brána pro připojení serveru a spouští definované spustitelné soubory napsané v jakémkoliv programovacím jazyce, přičemž vyžaduje specifický výstup skriptu (klíče / hodnoty a definici grafu). Díky tomuto principu je jednoduché vytvořit pluginy pro sběr nejrůznějších dat. Pro monitoring platformy byly vyvinuty a upraveny následující pluginy:

- ZooKeeper plugin (vytvořen, sběr dat z TCP streamu, Python)
- Solr plugin (vytvořen, sběr dat z REST API, Python)
- Tomcat plugin (modifikován pro Tomcat 7 s APR konektory, Perl)
- XWiki plugin (vytvořen, jednoduchý plugin pro zjištění stavu platformy, Bash)
- MySQL plugin (modifikován, přidány XtraDB specifické grafy, např. mutex monitoring, Bash a Perl)

Z nasbíraných dat je možné zjistit výkonnostní nedostatky, například pomalé disky, nedostatek paměti, zatížení *JVM heap*, či návštěvnost. Dále lze například detekovat problémy se synchronizací databáze a síťové problémy. Munin také nabízí možnost spustit libovolný skript při detekci chyby (pro sbíraná data lze nastavit limity), díky čemuž lze jednoduše nastavit emailové notifikace v případě vzniklé chyby. Pro co nespolehlivější monitoring byl monitorovací server umístěn na jiný virtuální server v jiné lokalitě. Ukázky grafů lze nalézt na obrázcích 3, 4 a 5.

5 Praktická část

Vytváření nového prostředí lze rozdělit na tři etapy:

1. Instalace middleware
2. Instalace a konfigurace XWiki
3. Úprava a vývoj funkcionalit

5.1 Instalace middleware

Celá platforma je nainstalována na třech identických serverech. Zvolen byl operační systém Red Hat Enterprise Linux z důvodu profesionální podpory v případě problémů s Linuxovým jádrem. Pro plánované budoucí použití *in-memory rendering cache*[7] bylo zvoleno 16GB operační paměti, 9GB bylo přiděleno aplikačnímu serveru pro XWiki a Solr. Část zbylé paměti byla přidělena databázovému serveru pro memory cache (InnoDB Buffer Pool), zbytek je použit pro podpůrný middleware a operační systém.

Pro potřeby clusteru byly alokovány čtyři IP adresy, jedna pro každý ze serverů a jedna IP adresa, která bude používána jako tzv. plovoucí.

Veškerý middleware byl instalován do speciálně vytvořeného LVM oddílu, který je připojen na */opt* mountpoint. Toto bylo zvoleno pro zamezení zahlcení systémového disku nepotřebnými daty (v době psaní této práce zabírá tento diskový oddíl 10GB, z toho 5GB databáze), LVM je použit z důvodu snadného rozšíření v případě nedostatku místa.

5.1.1 Konfigurace systémů

Základní instalace operačního systému RHEL 6 obsahuje spoustu zbytečných balíčků. Prvním krokem byla revize instalovaných balíčků a odinstalování nepotřebných. Kromě Xorg serveru a implementace OpenGL Mesa byly odstraněny i rozličné ovladače Wi-Fi karet, QLogic úložišť a IPTV karet. Taktéž byl odstraněn NetworkManager, jelikož nastavení sítě bude statické.

Následujícím krokem byla deaktivace SELinux, bezpečnostního rozšíření Linuxového jádra, z důvodu možného blokování instalovaného middleware. Taktéž byly aplikovány nejnovější bezpečnostní aktualizace.

5.1.2 Clusterware

Následně byl instalován a konfigurován zvolený clusterware, tedy Corosync a Pacemaker. Tyto aplikace jsou nabízeny v rámci předplatného RHCS a nejsou k dispozici z oficiálního RedHat repozitáře, lze je nicméně nainstalovat z repozitáře společnosti SUSE či z repozitáře distribuce CentOS.

[7] Uchovávání vykreslených stránek v operační paměti

V době instalace clusterware aplikací byla k dispozici stabilní verze Corosync 1.4, který využívá OpenAIS stack pro správu nastavených služeb a propojení s aplikací Pacemaker je zajištěno zastaralým pluginem. Ve verzi 2.X Corosync využívá všech výhod nového stacku *CMAN*. Pro konfiguraci služeb ve stávajícím prostředí je použit *crmsh* wrapper, v novější verzi je nahrazen modernějším *pcs*. Z důvodu možné interference mezi rozličnými cluster službami v použité sdílené podsíti byl Corosync nakonfigurován na UDP unicast. Tímto zabráníme dynamickému přidání nového serveru do stávajícího clusteru, kdy je třeba modifikovat konfiguraci všech existujících serverů a provést restart služby. Nastavení unicast komunikace nicméně přináší lepší bezpečnostní hledisko, jelikož nelze komunikaci jednoduše odposlechnout. Pro zvýšení bezpečnosti je komunikace napříč clusterem šifrovaná pomocí generovaného klíče o velikosti 1024 bitů[8].

Při startu operačního systému je spušten Corosync a Pacemaker. Jakmile je získáno *quorum* (je ustanoven vůdce clusteru), jsou spuštěny požadované služby. V případě, že server z clusteru vypadne, je aplikováno pravidlo *STONITH* a odpojený server násilně ukončí spravované služby pro vyvarování se možné situaci split-brain. I proto je vhodné server provozovat minimálně ve variantě se třemi instancemi. Kromě služeb nastaveny v režimu *cloned* mají všechny *active-passive* služby nastaveny tzv. **collocation**, neboli propojení. Tímto docílíme, že v případě neopravitelné chyby jedné z active-passive služeb budou všechny takto spojené služby automaticky přesunuty na server jiný. Pokud tedy vznikne například síťový problém a plovoucí IP adresa nebude moci být přidělena na určitý server, všechny ostatní služby budou přesunuty na server jiný. Je tedy vhodné takto propojit služby síťového stacku a softwarový loadbalancer.

5.1.3 Nginx

Nginx je použit jako softwarový loadbalancer a jednotný přístupový bod pro koncové uživatele. Také je použit pro zpracování SSL požadavků. Společně s použitím plovoucí IP adresy vznikne izolace fyzických serverů a skrytí skutečné infrastruktury před návštěvníkem. Lze také dynamicky upravovat nastavení HTTP protokolu, limity návštěv, či odstavit určitý server pro případ údržby, což by v případě použití DNS loadbalancingu nebylo možné.

Nginx využívá funkcionality vláken a víceprocesorového systému. Při inicializaci spustí několik procesů (každý může být odbavován jiným procesorem, v závislosti na schopnostech OS) a každý proces může vytvořit maximálně nastavený počet vláken. Je tedy možné současně efektivně odbavovat až P*V[9] klientů. Pro optimalizaci přenosu dat na transportní vrstvě jsou povoleny direktivy *sendfile*, *tcp_nopush* a *tcp_nodelay*, které zajistí optimálně velké pakety a minimalizují zpoždění způsobené *Nagleovým algoritmem*[5]. Následně je také nastavena *gzip* komprese pro textové dokumenty (*text/plain, text/css, application/x-javascript, text/xml, application/xml, application/xml+rss, text/javascript*). Tyto soubory jsou při požadavku zkomprimovány přímo na serveru a ke klientovi je přeneseno menší množství dat. Pro následnou cache těchto souborů je použita nginx

[8] Napevno nastaveno v aplikaci, bylo by vhodnější zvolit větší klíč

[9] počet procesorů * počet vláken

microcache, která uloží nastavené požadavky do paměti a je tedy tyto soubory možné odbavit rychleji. Toto nastavení je vhodné pro statické soubory, tedy obrázky, kaskádové styly, či JavaScript soubory a volitelně i pro dynamický obsah. Vzhledem k použití pouze v interní síti je zvýšen limit maximální velikosti požadavku na 2GB.

Nginx podporuje nastavení několika rozdílných webových adres, což plně podporuje protokol HTTP 1.1 (díky vynucení hlavičky *Host*). Pro možnost nastavení více SSL certifikátů podporuje nginx také SSL rozšíření *SNI*. U spojení je šifrování prováděno na nižší vrstvě OSI/ISO modelu, což znemožňuje identifikovat žádanou webovou URL pomocí zmíněné hlavičky. Pomocí metody SNI klient předem oznámí serveru o jakou webovou stránku má zájem a následná výměna klíčů již probíhá pomocí privátního klíče správného certifikátu.

Je zde nastaveno několik webových adres:

- XWiki (SSL, pro přímý přistup k platformě)
- Blog (SSL, pro přímý přístup k aplikaci na platformě XWiki)
- CRM (přístup k statickým informacím o clusteru, jiný port)
- Status (přístup k informacím o nginx serveru pro Munin monitoring, dostupné pouze z daného serveru na portu 81)

Pro zvýšení bezpečnosti je při přístupu na platformu přes nešifrovaný HTTP protokol poslán stavový kód 301 společně s hlavičkou Location signalizující přesměrování na zabezpečený kanál. Při předávání dat přes zabezpečený kanál je automaticky odeslána klientovi hlavička *Strict-Transport-Security*, díky které bude prohlížeč preferovat tento bezpečnější kanál.

Pro zabezpečení SSL přenosu jsou zakázány nebezpečné verze protokolu SSL a je povolen pouze protokol TLS. Dále jsou povoleny pouze dostatečně bezpečné šifry, jsou předgenerovány Diffie-Hellman parametry[6] a v případě široce dostupného webu je vhodné povolit i dodatečnou kontrolu platnosti certifikátu, tzv. *OCSP Stapling* a *Public Key Pinning*[7], který nastaví platné klíče certifikátů. Zamezí se tedy použití neautorizovaného certifikátu vydaného kompromitovanou důvěryhodnou certifikační autoritou jako například v případě CNNIC[8, 9]. Pro certifikát je použit privátní klíč o velikosti 2048 bitů.

Pro správné generování adres aplikačním serverem je vhodné nastavit HTTP hlavičky *X-Forwarded-For*, *X-Forwarded-Proto* a *X-Real-IP*, aby aplikační server dokázal správně detekovat IP adresu návštěvníka (*REMOTE_ADDR* proměnná je generována na straně aplikačního serveru). Na straně aplikačního serveru je třeba správně zpracovat přijímané hlavičky, pro Tomcat existuje built-in J2EE valve (ventil) *org.apache.catalina.valves.RemoteIpValve*.

5.1.4 Percona XtraDB Cluster

Percona XtraDB Cluster je nejvhodnější nainstalovat z oficiálního repozitáře[10] poskytovaného firmou Percona, který nám zajistí přístup k nejnovější stabilní verzi. Základní

[10]Centrální úložiště instalačních souborů

instalace vytvoří soubor */etc/my.cnf* obsahující konfiguraci a složku */var/lib/mysql/* obsahující databázová data. Prvním krokem bylo přesunutí této složky do adresáře */opt/xwiki/* z důvodu jistého nárůstu dat, změna byla třeba provést i v uvedeném konfiguračním souboru.

V konfiguračním souboru byly provedeny pouze malé změny, jelikož nelze odhadnout, které parametry je třeba upravit. Nastaven byl limit memory cache InnoDB (*innodb_buffer_pool*) na hodnotu 2GB pro cache častých dotazů a *performance-schema-instrument* pro dodatečný monitoring mutexů. Dále byly nastaveny potřebné parametry clusteru pro úspěšné propojení v rámci všech instancí. Zde byl nastaven název aktuálního serveru, název vytvořeného clusteru, údaje k databázi a doporučené nastavení[10] včetně *xtrabackup* pro *State Snapshot Transfer*[11]. Je vhodné nastavit i **gcache** pro případ výpadku serveru[11], zde nicméně díky problému s použitou verzí VMware hypervizoru (nemožnost alokovat velké soubory) nebyla použita.

Dále byl nakonfigurován a upraven odpovídající plugin do Munin monitoringu pro potřebná data v případné zjišťování výkonnostních problémů (*deadlock*, *slow queries*, aj.). Zde již je databáze dostatečně nakonfigurována a je ji třeba spustit. Vzhledem k neexistující Primary Component (zvolený vůdce clusteru) je třeba jeden server spustit jako vůdce příkazem *service mysql bootstrap-pxc*. Nyní již máme plně funkční databázový multi-master cluster.

5.1.5 LibreOffice

Dalším krokem byla instalace LibreOffice, který bude spuštěn v headless režimu jako server. LibreOffice je používán pro konverzi kancelářských dokumentů, je přes něj tedy možné efektivně importovat a exportovat například dokumenty s příponami DOC a XLS. Tímto docílíme užitečné funkcionality a umožníme uživatelům vytvářet hotové stránky z existujících dokumentů či generovat tabulky dle přílohy. Je také realizováno generování obrázků z prezentačních formátů pro uživatelsky přívětivé zobrazení a export stránek do formátu RTF a PDF. Pro propojení je používán *JODConverter*, fungující jako most mezi aplikací a LibreOfice serverem. Zvolen byl LibreOffice ve verzi 4.3, při testování verze 4.4 vznikaly problémy díky změně interního API. Pro správu serveru byl vytvořen Pacemaker agent za použití aplikace *screen*.

5.1.6 Apache Tomcat

Jelikož platforma XWiki i vyhledávací aplikace Solr figurují jako webové aplikace nevyžadující EJB, zvolen byl Tomcat jakožto nenáročný a rychlý aplikační server. Jak již bylo zmíněno v kapitole 4.3.1, byl nastaven s podporou APR konektoru (manuální kompilace APR a Tomcat-natives). Pro správu spuštěné instance byl vytvořen systémový *init skript*[12], kterého využívá i Pacemaker agent. Zde byla provedena základní konfigurace, modifikace portu serveru, smazání nepotřebných dat a nastavení autentikátoru pro Munin plugin (sběr informací o JVM, zatížení a přenesených datech s volitelnou možností

[11] Plná synchronizace dat
[12] Spouštěcí skript

monitoringu přes JMX). Dodatečná konfigurace byla provedena při konfiguraci platformy XWiki.

5.1.7 Solr a ZooKeeper

Solr je webová aplikace nabízející indexaci strukturovaných dat a jejich následné vyhledávání. Základním prvkem je *Core*, jádro, které obsahuje informace o přijímaných datech a jejich zpracování. XWiki nabízí možnost spustit Solr jako embedded (spuštěný při startu platformy pod vestavěným aplikačním serverem Jetty), nenabízí nicméně možnost clusterování. Z tohoto důvodu a z důvodu lepší správy byl zvolen *Solr Cloud*.

Solr Cloud využívá aplikace ZooKeeper, která je součástí projektu Hadoop a již řeší všechny cluster problémy. ZooKeeper funguje jako úložiště konfigurací a Solr zde má uložené definice jader a informace o existujících instancích. Každý Solr node byl nastaven na komunikaci se všemi ZooKeeper servery, aby se docílilo plné vysoké dostupnosti. V případě výpadku jednoho ze ZooKeeper serverů je tedy stav clusteru nezměněn a vyhledávání je stále funkční. Nákres infrastruktury je viditelný na obrázku 6.

5.2 Instalace a konfigurace XWiki

XWiki je webová aplikace distribuována jako *WAR* archiv. Složka */WEB-INF/* obsahuje konfigurační soubory. nejdůležitější z nich jsou **hibernate.cfg.xml**, **xwiki.cfg**, **xwiki.properties** a **observation/remote/jgroups/tcp.xml**. Volitelně lze nastavit Infinispan Cache v adresáři *cache/inifinispan/*.

Soubor *hibernate.cfg.xml* obsahuje nastavení databáze a connection pooling. Pro připojení k databázi byl zvolen oficiální MySQL JDBC konektor, jakožto stabilní a výkonná implementace. Bylo zváženo i použití MariaDB konektoru, jelikož Percona XtraDB je základní databázový engine distribuovaný s MariaDB, nicméně z důvodu nedokonalosti a nestabilitě nebyl zvolen. Pro connection pooling byl zvolen HikariCP, který musel být zkompilován s vlastní implementací *Hibernate Connection Provider* vzhledem k použité starší verzi JPA implementace Hibernate. Zde bylo použité následující nastavení: 20 spojení minimum, 50 spojení maximum, timeout 30 sekund.

V konfiguračním souboru *xwiki.cfg* je základní nastavení platformy. Nejdůležitějším nastavením zde je ukládání dat, kdy XWiki v základním nastavení ukládá veškeré přílohy do databáze. Vzhledem k nevhodnosti ukládání binárních dat do relační databáze bylo ukládání příloh nastaveno na sdílený systém souborů (NAS). Dále se zde nastavuje například povolení databázové migrace při aktualizaci platformy na novější verzi, cache, či nastavení autentizace. Zde byla také nastavena vlastní implementace autentikátoru pro *SSO*.

Konfigurační soubor *xwiki.properties* obsahuje konfigurační data použitých komponent. Nejdůležitějším parametrem je nastavení adresáře pro ukládání permanentních dat (přílohy a logy komponent). Dále zde je třeba nastavit OpenOffice a Solr komponenty pro použití externích instancí a *Observation modul*. Observation modul je API tvořící most mezi event-driven principem XWiki a JGroups knihovnou pro zajištění synchronizace v

clusteru. Vzhledem k aktivní Infinispan cache by v případě vzniklé změny ostatní instance nebyly informovány a například při úpravě dokumentu by změna nebyla viditelná. Na toto slouží právě JGroups, který zajišťuje přenos vzniklých událostí (eventů) mezi ostatními instancemi. V *xwiki.properties* je třeba tuto funkcionalitu zapnout a nastavit cestu k JGroups konfiguračnímu souboru (tcp.xml zmíněný v úvodu kapitoly). Zde byla zvolena metoda *TCPU* (TCP unicast), JGroups tedy komunikuje pouze se servery, které má nastavené v daném konfiguračním souboru. Je zde tedy myšleno na stejné bezpečnostní hledisko jako v případě aplikace Corosync. Pokud by daný cluster byl větší, ve vzdálenějších lokalitách, nebo by vzniklých událostí bylo hodně, je vhodnější zvolit UDP unicast.

WAR archiv XWiki dále obsahuje složky */resources/* obsahující permanentní statické knihovny třetích stran (JavaScript a CSS, zde se přechází na použití **webjars**), */templates/* obsahující základní šablony jednotlivých akcí a */skins/* obsahující jednotlivé vzhledy XWiki. V novější verzi (6.0+) se používá skin **flamingo**, který využívá schopností knihovny **Bootstrap** a poskytuje moderní vzhled.

Zde je již instance XWiki v základní konfiguraci, dále bylo třeba provést změny týkající se vzhledu dodržujícího interní UI standard a integraci s používanými službami.

5.3 Úprava a vývoj funkcionalit

5.3.1 Single Sign On

Ve firmě se používá autentizace s pomocí *Active Directory*, nejvhodnější je tedy použít určitou formu Kerberos SSO díky použití Linux serveru. Je důležité zmínit použití více AD serverů (KDC), je tedy třeba použít implementaci, která toto umožňuje. Zde byla zvolena metoda *SPNEGO*, nahrazující zastaralé a nebezpečné *NTLM*, konkrétně knihovna SPNEGO SourceForge poskytující jednoduchý přístupový bod k *JAAS*. Tato knihovna obsahuje *J2EE filter*, který lze jednoduše použít v konfiguračním souboru použitého aplikačního serveru. Tímto docílíme pouze autorizovanému přístupu k platformě.

Zde nicméně vzniká několik problémů. V případě kontroly dostupnosti je třeba mít Kerberos ticket a klienta s podporou GSSAPI. Zde se dá použít knihovna *libcurl*, obsahující podporu zmíněného API, nicméně se nelze spoléhat na použití této knihovny všemi aplikacemi. SPNEGO knihovna již obsahuje funkcionalitu pro obejití autentizace v případě návštěvy z lokálního počítače (volitelně), byla tedy implementována funkcionalita pro pevné přiřazení uživatelského jména specifikované IP adrese. Následně zde byl nalezen problém, který lze jasně vyvodit ze specifikace protokolu. V případě návštěvy stránky je přístup zamítnut a klientovi je odeslána hlavička žádající o autentizaci. Klient následně odešle požadavek znovu a připojí hlavičku, pomocí které server ověří validitu údajů. Klient tedy požadavek odesílá dvakrát, což v normálním případě nezpůsobuje žádný problém, nicméně v případě odesílání multipart dat (soubory) jsou tyto soubory odeslány dvakrát, což způsobuje jednak uživatelskou nepřívětivost, tak i zvýšenou zátěž serveru. Pro řešení tohoto problému se nabízejí dvě možnosti:

1. Speciální stránka pro autentizaci

2. Příznak specifikující již proběhlou autentizaci (funkcionalita *sticky-cookies*)

První možnost znamená vytvoření speciální stránky, která zpracuje přihlášení a nastaví příznak, že autentizace proběhla (stejně jako v bodě č. 2). Výhodou a zároveň i nevýhodou této metody je, že autentizace je pouze "volitelná", musíme tedy počítat s anonymním přístupem na platformu.

Tento problém řeší metoda č. 2, kdy byla do SPNEGO knihovny implementována funkcionalita umožňující propuštění požadavku v případě nalezené *cookie*. Zde bylo třeba vytvořit bezpečnou implementaci ověření v autentikátoru na straně XWiki, aby nedošlo k propuštění neplatného požadavku, například v případě modifikace hodnoty cookie. V případě první návštěvy platformy je klient autentizován pomocí knihovny SPNEGO a verifikován na straně XWiki. Zde je vytvořen speciální řetězec ve formátu *"username@REALM:timestamp"*, zašifrován silnou symetrickou šifrou a nastaven jako hodnota potřebné cookie. V případě opakované návštěvy klienta je prohlížečem odeslána tato cookie, SPNEGO autentikátor propustí požadavek bez verifikace platnosti a požadavek je zpracován XWiki autentikátorem. Ten se pokusí hodnotu cookie rozšifrovat a ověří její platnost (volitelně i *timestamp*[13]) a v případě neplatnosti cookie smaže a nastaví přesměrování. Zde již klient cookie nepošle a je autentizován SPNEGO knihovnou.

V obou případech je třeba myslet na několik bezpečnostních hledisek, například délka platnosti cookie, modifikace cookie klientem, zcizení cookie, či nedostatečně silná šifra. Vzhledem k použití v korporátní sféře a nedostupnosti platformy přes internet bylo předpokládáno, že zcizení cookie není pravděpodobné a šifra je dostatečně silná.

Dále bylo v knihovně opraveno několik bezpečnostních problémů a přidána možnost zobrazit klientovi volitelnou přihlašovací zprávu pomocí parametru *realm* u *Authorization* hlavičky HTTP protokolu. Problémem také bylo, jak zajistit zobrazení informací uživateli v případě, že se nebude moci přihlásit (nedostupné KDC, chybné údaje, či jakákoli jiná chyba). Pokud uživatel odmítne záložní *Basic autentizaci*[14], odpověď skončí s HTTP chybou 401, stejnou chybu je ale nutné posílat i v případě, že autentizace probíhá (pro správnou Negotiate autentizaci je třeba vykonat dva HTTP požadavky), není tedy možné tento "chybový"stav odchytávat na straně proxy serveru. Taktéž není žádoucí posílat velký statický obsah v těle požadavku z důvodu zvýšené zátěže a zhoršené možnosti modifikace (nutný restart aplikačního serveru). Jelikož chybový kód není ze sekce přesměrování (stavy 3xx), prohlížeč by jakýkoliv pokus o přesměrování s pomocí hlavičky *Locate* ignoroval. Je nicméně možné zkombinovat možnost odeslání těla požadavku při stavu 401 s možností jazyka HTML (atribut **http-equiv** u tagu *meta*). Použitím zmíněného atributu uvnitř tagu meta signalizujeme prohlížeči, že máme zájem o přesměrování, které započne až v případě načtení celé stránky. Pokud se uživatel úspěšně autentizuje, tělo odpovědi je zahozeno, pokud uživatel autentizaci odmítne, bude přesměrován na námi požadovanou stránku. Odesílaná zpráva má malou velikost, umožňuje dynamicky upravovat odkazovanou stránku a v případě kombinace se sticky-cookies je zvýšená zátěž minimální. Původní verze knihovny SPNEGO tuto funkcionalitu neumožňuje, byla tedy implementována.

[13] Časové razítko

[14] Přihlášení jménem a heslem

Následujícím problémem byla konzistence uživatelských dat (osobní data, informace o skupinách). Možnosti existují dvě, synchronizovat uživatelská data synchronně při návštěvě (autentizaci) a asynchronní synchronizace všech uživatelů. Zvolena byla druhá možnost vzhledem k neintruzivnímu použití. Byla vytvořena komponenta která každých 24 hodin synchronizuje osobní informace a informace o skupinách všech uživatelů ze všech dostupných KDC serverů. Pro možnost clusterování bylo nastavené počáteční zpoždění (náhodně generované) a vytvořen nový event *LDAPGroupSynchronizationEvent*. V případě spuštěné synchronizace na jedné instanci je tento event vypálen, pomocí JGroups je distribuován skrze všechny spuštěné instance. Komponenta dále obsahuje event listener, který zachytí vypálený event, zruší vytvořený časovač a vytvoří nový se zpožděním 24 hodin + náhodný interval. Tímto docílíme synchonizace pouze na jedné instanci a v případě výpadku bude synchronizace spuštěna na jiné instanci.

5.3.2 Úprava vzhledu

Pro dodržení interních UI standardů byla lehce upravena základní šablona Flamingo a vytvořena speciální stránka obsahující SkinExtensions, které se načítají u všech stránek. Bylo tedy dosaženo efektivní a snadné změny vzhledu. Bylo změněno i rozložení úvodní stránky pro snadnou orientaci mezi dostupnými stránkami.

5.3.3 Blogy

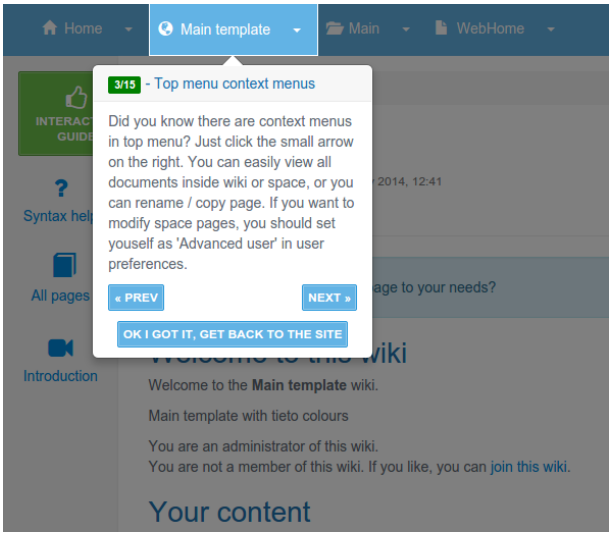
Pro potřebu blogování byla použita Blog aplikace distribuována s XWiki. Aplikace byla upravena a byla přidána funkcionalita pro interní potřeby (nahrávání obrázků jednotlivým příspěvků s využitím funkcionalit technologie HTML5). Aplikace je dostupná na oddělené URL.

5.3.4 WYSIWYG Editor

XWiki obsahuje propracovaný a snadno rozšiřitelný WISYWYG editor, který nabízí spoustu možností. WYSIWYG editor samotný je vytvořen pomocí *GWT*, knihovny vyvinuté společností Google pro jednoduché tvoření dynamických webových aplikací. Samotná aplikace je napsána v programovacím jazyce Java, využívající veškerých výhod tohoto jazyka, a následně je zkompilována pomocí GWT do jazyka JavaScript. Tento editor nicméně není dokonalý a bylo ho potřeba lehce upravit. V případě, že chcete změnit vlastnosti vloženého obrázku, je třeba obrázek nejdříve označit a posléze kliknout na příslušnou ikonku v menu. Editor byl tedy upraven s funkcionalitou dvojkliku, tedy v případě dvojkliku na obrázek bude otevřeno okno modifikací. Interně bylo tedy nutné vytvořit event listener na *DoubleClickEvent*, získat daný element, označit ho a spustit otevření modifikačního okna (výpis 1).

Další modifikací bylo volitelné vytvoření čisté HTML tabulky nepoužívající základní vzhled.

```
/**
 * {@inheritDoc}
```



Obrázek 1: Úvodní dynamický tutoriál na XWiki

```
 *
 * @see DoubleClickHandler#onDoubleClick(DoubleClickEvent)
 */
public void onDoubleClick(DoubleClickEvent event)
{
    Element elem = Element.as(event.getNativeEvent().getEventTarget());
    if (elem != null && event.getSource() == plugin.getTextArea()
            && "img".equalsIgnoreCase(elem.getTagName())) {
        Document doc = plugin.getTextArea().getDocument();
        Selection s = doc.getSelection();
        Range r = doc.createRange();
        r.selectNode(elem);
        s.removeAllRanges();
        s.addRange(r);
        plugin.onImageEdit();
    }
}
```

Výpis 1: Zpracování dvojkliku na obrázek v GWT

5.3.5 Úvodní dynamický tutoriál

Pro nezkušené uživatele byl vytvořen dynamický tutoriál pomocí JavaScript knihovny *Bootstro.js*, která využívá funkcionality *Popover* knihovny Bootstrap. Tímto bylo dosaženo uživatelsky přívětivě prezentace jednotlivých prvků na XWiki.1

5.3.6 Podpůrné komponenty

XWiki v základu obsahuje komponentu pro autentizaci vůči LDAP serveru. Této komponenty nebylo využito, byla nicméně základem při tvorbě vlastní komponenty pro periodickou synchronizaci uživatelských dat. Umožňuje důležitou funkcionalitu, synchronizaci uživatelských skupin se skupinami na XWiki. Při návrhu funkčnosti byla zvolena metoda inspirována návrhovým vzorem *Lazy loading*, tedy každý zaměstnanec se stane uživatelem wiki až v případě, že jí navštíví. Tato volba byla zvolena z důvodu menší zátěže. V případě, že se vytváří nová týmová subwiki, či je třeba nastavit oprávnění uživateli, je nutné uživatele požádat, zda by mohl nejdříve platformu navštívit. Toto řešení je neefektivní a zdlouhavé, byla tedy přidána funkcionalita do synchronizační komponenty, dostupná uživatelům s administrátorským oprávněním, která dokáže simulovat přihlášení uživatele a vyžádá synchronizaci potřebných informací.

Pokud je třeba nastavit zmíněnou synchronizaci uživatelských skupin, je nutné vložit přesný unikátní identifikátor (DN) nacházející se v LDAP serveru. Jelikož k této informaci nemají většinou koncoví uživatelé přístup, byla do zmíněné komponenty přidána také možnost zobrazit všechny skupiny daného uživatele. Poslední funkcionalitou této komponenty, kromě manuálního spuštění synchronizace všech uživatelů, je spuštění synchronizace specifického uživatele v případě nutné akutní změny informací (a především zařazení do skupin).

Díky použití centralizované správy uživatelů pomocí Active Directory je uživatelský obrázek uložen ve formátu čistých binárních dat v atributu *jpegPhoto*. Přestože z názvu atributu by bylo patrné, že obrázek bude ve formátu JPEG, není to nutností a je třeba počítat i s jiným formátem obrázků (z binárních dat je třeba zjistit správný tzv. *content type*). Jelikož nebylo vhodné ukládat obrázky jako přílohu k jejich stránce z důvodu verzování a zvýšené zátěže, byla vytvořena oddělená J2EE aplikace, která periodicky stahuje aktuální uživatelské obrázky z LDAP serverů a ukládá je na disk ve formátu *uzivatelske_jmeno.pripona*. Data jsou přístupná přes server nginx, je tedy jejich odbavení rychlé a nezatěžuje aplikační server. Zde bylo využito znalostí z předmětu **Java Technologie** a aplikace využívá CDI a JNDI.

5.3.7 Nalezené problémy

V produkčním provozu platformy bylo nalezeno několik podstatných problémů, které musely být vyřešeny. Prvním problémem byla odezva databázové replikace. V případě rozsáhlejší změny trvala synchronizace delší časový úsek (v řádu sekund), zatímco JGroups již distribuoval vzniklé události. Následně došlo na ostatních instancích v clusteru k revalidaci cache na datové vrstvě, nicméně dokument nebylo možné z databáze načíst a revalidace skončila s chybou. Bylo tedy možné, aby se porušila synchronicita instancí a každá mohla prezentovat rozdílné informace. Zde bylo zprvu využito možné modifikace JGroups pomocí protokolu **DELAY**, kterým lze nastavit zpoždění odesílaných informací. Toto řešení nicméně problém neřeší, v případě práce s větším obnosem dat (například větší transakce při hromadné úpravě dokumentů) bylo možné se s uvedeným problémem setkat, naštěstí pouze v minimální míře. Následně byl zjištěn problém s imple-

mentací uvedeného protokolu, kdy každý požadavek je spouštěn v odděleném vlákně a vlákno je zablokováno po dobu uvedenou při konfiguraci protokolu. Je tedy teoreticky možné, aby byl server neúměrně zatížen při vzniku více událostí po danou dobu (1).

> *bela_: Note that this blocks the sending thread, and uses a receiving thread*
> *bela_: Pretty simple&stupid impl, I should have used tasks to free the thread and deliver the message later*
>
> (1)
>
> ***Bela Ban, autor JGroups, diskuze problémů s DELAY protokolem***
> *7. listopad 2014, #jgroups kanál na irc.freenode.net*

Zde byla nakonec zvolena modifikace JDBC nastavení. Použitý MySQL konektor umožňuje nastavit loadbalancing mezi více servery. Jelikož po delším zkoumání nebyla nalezena podrobná specifikace dané funkcionality, byla zvolena možnost jiná. Jelikož je garantováno, že na serveru obsahující plovoucí IP adresu je vždy dostupný databázový server, byl konektor nastaven pro připojení právě na tuto plovoucí IP adresu. V jednu dobu tedy všechny instance přistupují ouze na jeden databázový server, nedochází zde ale k porušení vysoké dostupnosti. V případě problému na daném serveru je plovoucí IP adresa přesunuta na server jiný a databázové spojení je znovu vytvořeno, dojde zde tedy ke minimální prodlevě, díky rychlému navázání spojení s databázovým serverem. Alternativním řešením by byla modifikace jádra XWiki, vytvoření fronty přijatých JGroups událostí a vícenásobným pokusům o zpracování událostí. Zde by nicméně bylo nutné řešit jistou transakční vrstvu.

Další nalezený problém souvisí s aplikací Blog, která funguje na jiné doméně. Jelikož XWiki umožňuje doménově oddělit pouze jednotlivé wiki, docházelo k problémům s CORS při AJAX požadavcích (například při nahrávání obrázků). Zde bylo prvním pokusem použít vestavěný Tomcat filtr, ten se nicméně ukázal jako nedokonalá implementace a problém neřešil. Byl tedy vytvořen vlastní filtr, který danou funkcionalitu implementoval a v případě nalezené správné *Origin* hlavičky byly nastaveny správné hlavičky (*Access-Control-Allow-Origin* pro povolení požadavku a *Access-Control-Allow-Credentials* pro odesílání cookies v požadavku). Jelikož je daná bezpečnostní funkcionalita kontrolou pouze na straně klienta, není třeba řešit neplatnou hlavičku Origin.

5.3.8 In-memory rendering cache

Dalším problémem byla implementace další vrstvy cache. Jelikož zpracování získaných dat může trvat delší dobu, například díky skriptování a pohledům přes více stránek, je vhodné implementovat cache pro vykreslená data. Výsledky porovnání rozdílných způsobu cache jsou zobrazeny v tabulce 5. XWiki již obsahuje jistou implementaci, nicméně ji bylo třeba nejdříve opravit, jelikož nepočítala s použitím SkinExtensions. Dále bylo třeba vytipovat stránky, které nemají danou cache používat, například rozdílné dynamické administrační stránky. Největším problémem nicméně bylo zpracování vztahů mezi strán-

kami. Jestliže je stránka upravena, je její cache vytvořena znovu, nicméně pokud na dané stránce závisí stránky jiné, je jejich cache nedotčena a je zobrazován starý obsah. Například pokud existuje týmová wiki zobrazující kalendář, kdy každá stránka odpovídá jednomu záznamu v kalendáři, v případě přidání, smazání, či úpravě záznamu je výpis o změnách neinformován a zobrazuje zastaralé informace. Jelikož je vhodné udržovat stránky v paměti co nejdelší dobu, vzniká zde zásadní problém, který je třeba vyřešit.

XWiki již obsahuje statistický modul, který schraňuje informace o vztahu jednotlivých stránek, monitoruje nicméně pouze určité závislosti (například makra pro zobrazení části jiné stránky). Tento modul je navíc poměrně náročný, byla tedy zvolena možnost vývoje vlastní komponenty. Modifikací rendering modulu v XWiki jádře byla přidána možnost zneplatnit cache specifikované stránky. Implementovaná komponenta poskytuje API použitelné při skriptování, které umožní uživateli definovat závislosti této stránky či stránku označit jako závislost stránky jiné. Také obsahuje metody pro zrušení závislostí a statistické metody pro administraci. Interní částí této komponenty je Singleton obsahující podstatnou implementaci mazání cache, EventListener pro události stránek (úprava, přidání, smazání) a řada "správců".

Princip komponenty spočívá v existenci tzv. kotev stránek, řetězci, které definují, že pro danou stránku existuje závislost. Použitá implementace je pomocí hašovacích tabulek pro rychlé nalezení potřebných informací. Díky použití CDI, kde existuje rozhraní a jeho implementace, je možné jednoduše vytvořit novou implementaci rozhraní s vyšší prioritou. Zde existuje několik případů užití:

1. Stránka je závislá na jiné stránce
2. Stránka je závislá na stránkách v jistém Space
3. Stránka je závislá na stránce o určitém názvu v jakémkoliv Space
4. Stránka je závislá na svém potomkovi
5. ...

Díky rozdílným případům užití a možnosti vzniku dalších byla zvolena modulární implementace pomocí zmíněných správců. Každou kotvu definuje třída implementující rozhraní *CacheObserverHandler*, které obsahuje metodu vracející seznam kotev pro daný dokument. Při registraci závislosti se uloží tato kotva jako klíč tabulky a při vzniku události je z definice všech kotev získán seznam kotev odpovídající danému dokumentu. Iterací tohoto seznamu a dotazu na uloženou strukturu je získán seznam dokumentů, kterým se má zneplatnit cache (výpis 2). Použitá struktura pro uložení dat, hašovací tabulka, obsahuje kotvu jako klíč a jako hodnotu množinu popisovačů stránek, aby nebyla držena reference na celý dokument a nebyla tedy ztížena práce garbage collectoru.

V případě, že chceme označit rodičovskou stránku *Test.Parent* jako závislost, stačí registrovat kotvu **parent:xwiki:Test.Parent**, kterou definuje daná implementace (výpis 3). V případě vzniklé události je vytvořena kotva **parent:Rodič.Stránky** a je vyhledána v použité struktuře. Zde je získán seznam závislých stránek a jejich cache je zneplatněna.

Tato implementace nabízí široké možnosti rozšiřování a efektivní správu závislostí zcela nezávislých na použitém propojení a použití je zcela v režii uživatele.

5.4 Testování nového prostředí

Díky neexistujícímu automatickému testování platformy byla zvolena manuální kontrola funkcionalit celé platformy. Pomocí nástroje *ab* byla vykonána série zátěžových testů na celou platformu a ze získaných dat z nástroje Munin byla vytvořena základní konfigurace jednotlivých komponent.

5.5 Migrace z jiných platforem

Důležitým úkolem po vytvoření platformy byla migrace dat z různorodých prostředí. Prvním migračním projektem byl import dat ze systému Trac. XWiki nabízí několik možných způsobu vložení dat. Nejpoužívanějším způsobem je XAR baliček. XAR je archiv serializovaných stránek do XML souborů a každou existující stránku je možné do tohoto formátu exportovat. Lze tak dosáhnout poměrně efektivní zálohy potřebných stránek.

Druhým způsobem importu dat je REST API nabízené platformou. Zde lze POST požadavkem poslat serializovaný dokument ve formátu XML na adresu, která specifikuje URL vytvářené stránky. Lze tak efektivně hromadně vytvářet stránky z existujících dat.

XWiki podporuje několik možných jazykových syntaxí od XWiki syntaxe, přes HTML až po Markdown. Systém Trac nicméně používá vlastní syntaxi, kterou platforma nepodporuje. Z tohoto důvodu byl použit skript vytvořený v jazyce Perl, který zpracovává gramatiku Trac jazyka a generuje MediaWiki syntaxi. Zde bylo třeba skript částečně upravit a s pomocí podpůrných skriptů v jazyce Python byly automaticky převedeny všechny potřebné stránky do XWiki syntaxe. Jelikož systém Trac umožňuje verzování stránek, bylo nutné vytvořit ekvivalentní počet XML souborů se specifikovanou prioritou importu.

6 Závěr

Má práce za dobu absolvování této praxe vedla k vytvoření vysoce dostupné platformy postavené na open-source technologiích, která funguje spolehlivě pro větší množství uživatelů a je efektivní náhradou některých proprietárních placených technologií. Vytvořená platforma je hojně používána napříč firmou a je užitečným nástrojem pro rozličné týmy.

V průběhu absolvování praxe jsem se setkal s řadou zajímavých a neobvyklých problémů z rozdílných odvětví informačních technologií. Výhodou byla jistá předchozí znalost dané problematiky a všeobecný přehled. Vzhledem k použití operačního systému Linux by bylo vhodné absolvovat předmět *Správa operačních systémů*, který byl nicméně k dispozici až pro poslední semestr bakalářského studia. Díky rozsáhlému použití platformy zde byl brán zřetel i na bezpečnost celého řešení, což je cennou zkušeností.

Výhodou také bylo absolvování předmětů *Java technologie*, *Úvod do databázových a informačních systémů*, *Skriptovací programovací jazyky* a *Operační systémy*, které byly přínosem při návrhu a vytváření platformy.

7 Reference

[1] *Writing XWiki Components.* XWIKI SAS. Documentation for the XWiki Platform: Developer's Guide [online]. 2014 [cit. 2015-04-11]. Dostupné z: http://platform.xwiki.org/xwiki/bin/view/DevGuide/WritingComponents

[2] *Skin Extension Tutorial.* XWIKI SAS. Documentation for the XWiki Platform: Developer's Guide [online]. 2014 [cit. 2015-04-11]. Dostupné z: http://platform.xwiki.org/xwiki/bin/view/DevGuide/SkinExtensionsTutorial

[3] *Scripting.* XWIKI SAS. Documentation for the XWiki Platform: Developer's Guide [online]. 2014 [cit. 2015-04-11]. Dostupné z: http://platform.xwiki.org/xwiki/bin/view/DevGuide/Scripting

[4] *Index of wsrep_provider options: pc.recovery.* PERCONA. Percona XtraDB Cluster 5.6 Documentation [online]. 2014 [cit. 2015-04-10]. Dostupné z: http://www.percona.com/doc/percona-xtradb-cluster/5.6/wsrep-provider-index.html#pc.recovery

[5] *Nginx optimization: Understanding SENDFILE, TCP_NODELAY and TCP_NOPUSH.* DE VILLAMIL, Frédéric. The UX Ray [online]. 2014 [cit. 2015-04-08]. Dostupné z: https://t37.net/nginx-optimization-understanding-sendfile-tcp_nodelay-and-tcp_nopush.html

[6] *Dhparam - DH parameter manipulation and generation.* OpenSSL developers. Official OpenSSL Documentation [online]. 2015 [cit. 2015-04-08]. Dostupné z: https://www.openssl.org/docs/apps/dhparam.html

[7] *Strong SSL Security on nginx.* VAN ELST, Remy. Raymii.org [online]. 2015 [cit. 2015-04-08]. Dostupné z: https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html

[8] *Maintaining digital certificate security.* LANGLEY, Adam. GOOGLE INC. Google Online Security Blog [online]. 2015 [cit. 2015-04-08]. Dostupné z: http://googleonlinesecurity.blogspot.cz/2015/03/maintaining-digital-certificate-security.html

[9] *Revoking Trust in one CNNIC Intermediate Certificate.* WILSON, Kathleen. MOZILLA SECURITY TEAM. Mozilla Security Blog [online]. 2015 [cit. 2015-04-08]. Dostupné z: https://blog.mozilla.org/security/2015/03/23/revoking-trust-in-one-cnnic-intermediate-certificate/

[10] *Installing Percona XtraDB Cluster on CentOS.* PERCONA. Percona XtraDB Cluster 5.6 Documentation [online]. 2014 [cit. 2015-04-10]. Dostupné z: http://www.percona.com/doc/percona-xtradb-cluster/5.6/howtos/cenots_howto.html

[11] *How to calculate the correct size of Percona XtraDB Cluster's gcache.* NIETO, Miguel Angelo. PERCONA. Percona MySQL Performance Blog [online]. 2014 [cit. 2015-04-10]. Dostupné z: http://www.percona.com/blog/2014/09/08/calculate-correct-size-percona-xtradb-clusters-gcache/

A Výpisy kódu

```
@Inject
private Provider<List<CacheObserverHandler>> cacheObserverHandlerProvider;

@Override
public Set<DocumentReference> obtainHandlers(XWikiDocument doc)
{
    Set<DocumentReference> handlers = new HashSet<DocumentReference>();

    for (CacheObserverHandler h: cacheObserverHandlerProvider.get()) {
        for (String fullname: h.getURIs(doc)) {
            logger.info("Searching for '{}' in tree with size {}", fullname, tree.size());
            Set<DocumentReference> tmp = tree.get(fullname);
            if (tmp != null) {
                logger.info("Obtained {} handlers for '{}' with document {}",
                        tmp.size(), fullname, doc.getDocumentReference());
                handlers.addAll(tmp);
            }
        }
    }
    logger.info("Obtained {} handlers for '{}' page", handlers.size(), doc.getDocumentReference
        ());

    return handlers;
}
```

Výpis 2: Metoda pro získání závislostí předané stránky

```
/**
 * Cache handler for parent document lookup (parent:subwiki:Space.Page).
 *
 * @version $Id$
 */
@Component
@Named("CacheObserverHandler_parent")
public class ParentHandler implements CacheObserverHandler
{

    @Override
    public List<String> getURIs(XWikiDocument doc) {
        return Arrays.<String> asList(
                String.format("parent:%s", doc.getParentReference().toString())
        );
    }

}
```

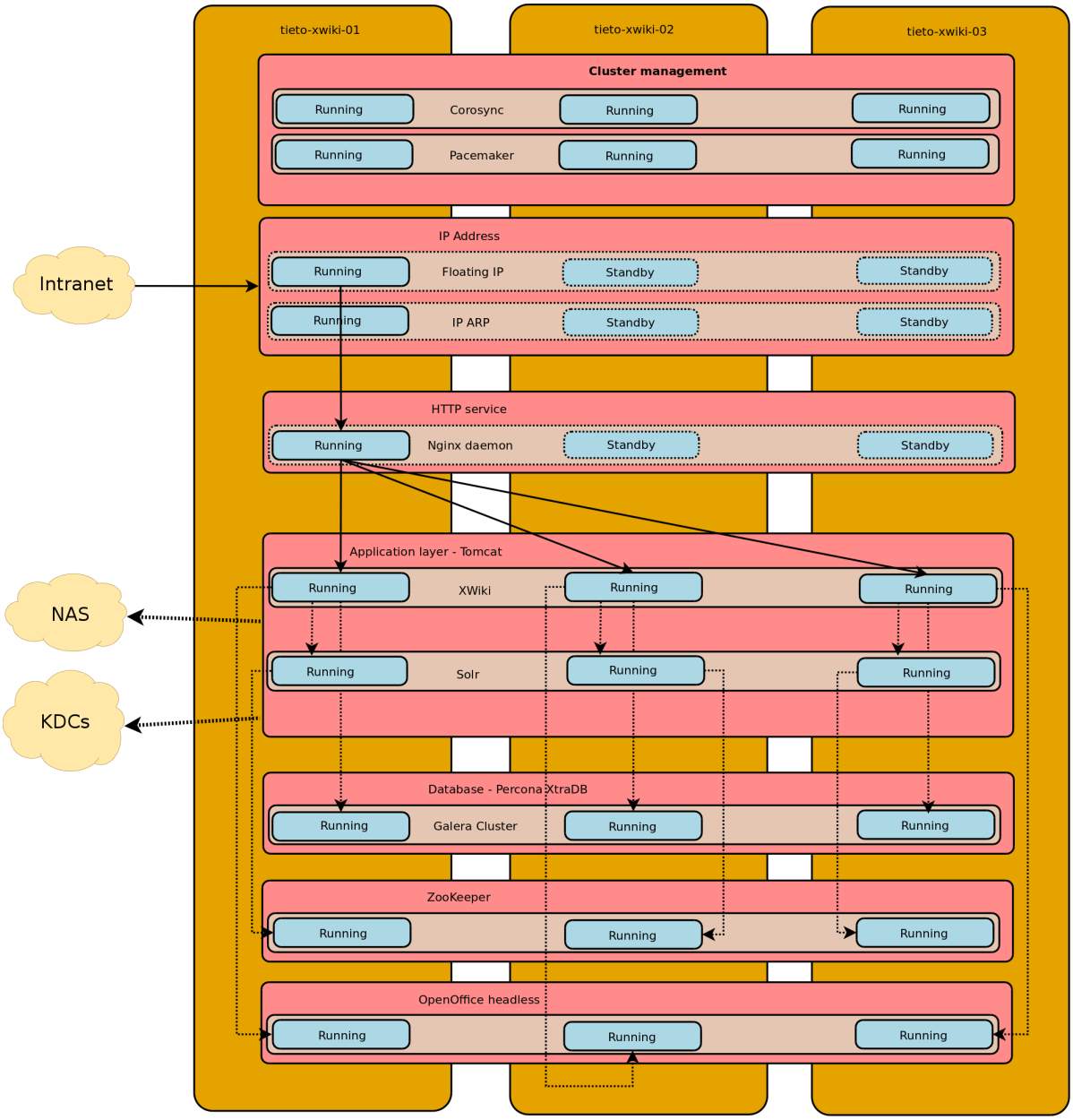
Výpis 3: Implementace kotvy pro definování rodiče

B Tabulky

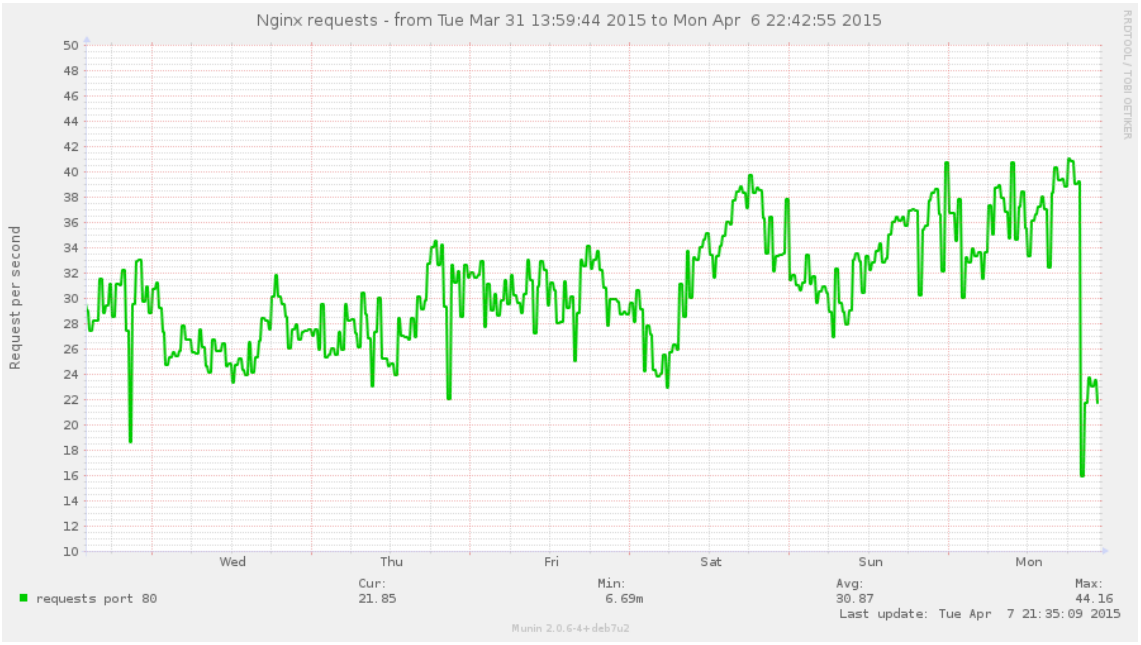
Typ stránky	Cache	Průměrná doba (ms)	První načtení (ms)
Neexistující bez UI	Infinispan	165.918	198
	Infinispan + rendering	163.114	
Existující bez UI	Infinispan	214.908	293
	Infinispan + rendering	168.443	
Existující s UI	Infinispan	384.090	905
	Infinispan + rendering	330.149	
Existující složitější s UI	Infinispan	541.222	13 376
	Infinispan + rendering	336.260	

Tabulka 5: Výkonnostní test XWiki

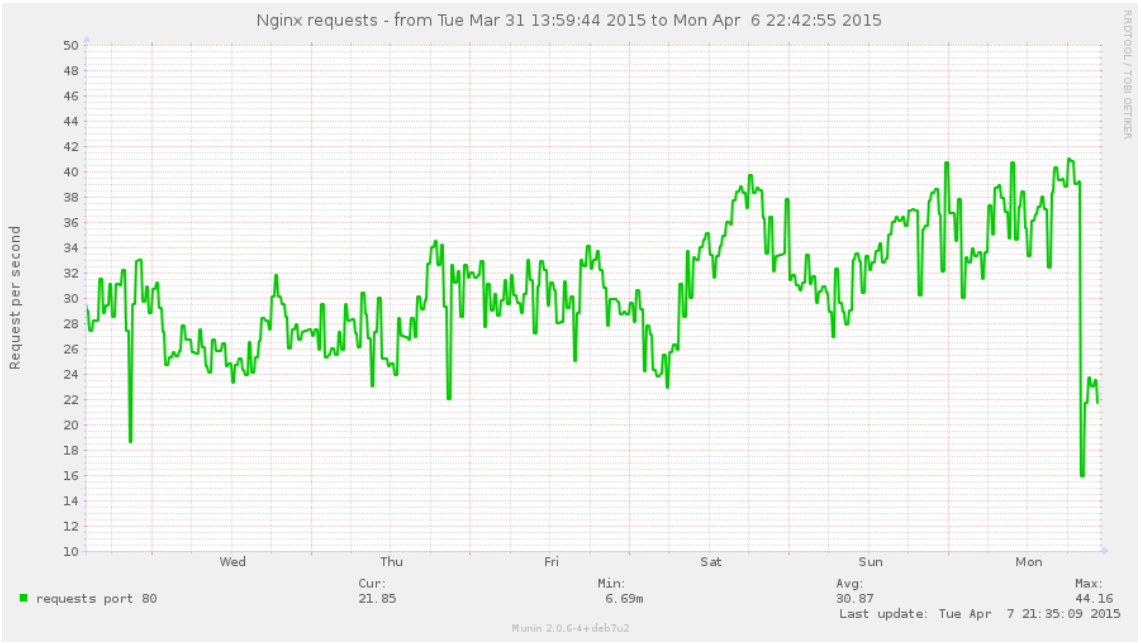
C Obrázky



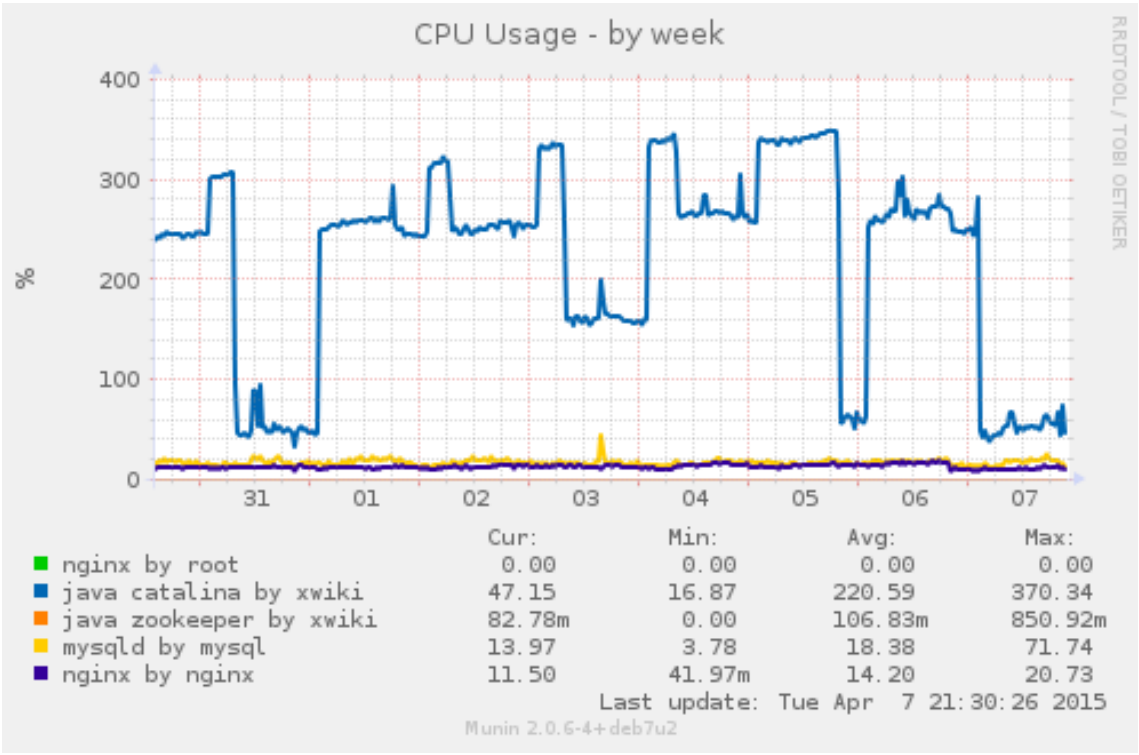
Obrázek 2: Navrhovaná nová infrastruktura



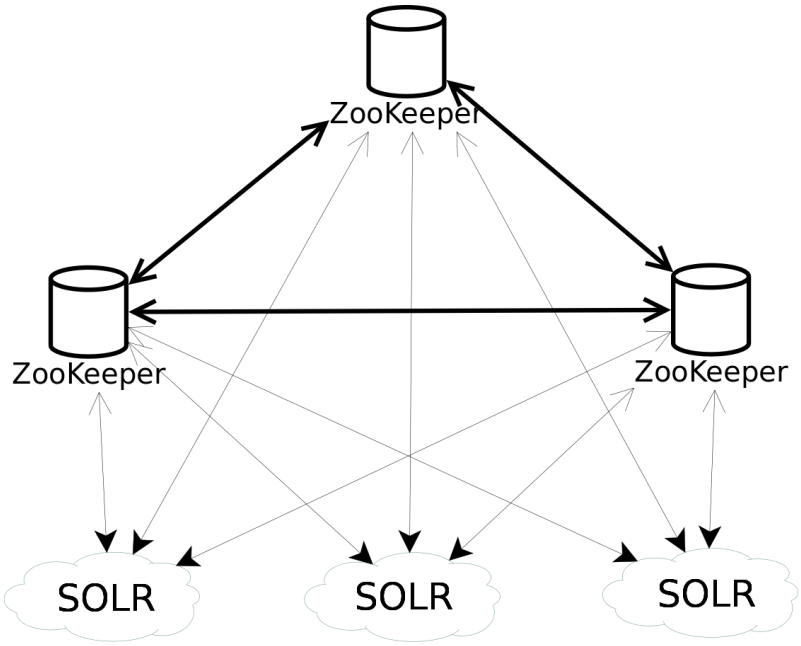
Obrázek 3: Munin graf, počet přístupů na nginx



Obrázek 4: Munin graf, počet přístupů na nginx, podrobnější



Obrázek 5: Munin graf - ukázka, zatížení procesoru



Obrázek 6: Nastavení Solr Cloud se ZooKeeper servery pro vysokou dostupnost